ELSEVIER

Future Generation Computer Systems 16 (2000) 379–391

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Privacy protection and anonymity services for the World Wide Web (WWW)

Rolf Oppliger *

Swiss Federal Office of Information Technology and Systems (BFI), Monbijoustrasse 74, CH-3003 Bern, Switzerland

Accepted 3 March 1999

Abstract

Although browsing on the World Wide Web (WWW) feels like an anonymous activity, it is hardly that way. Website administrators generally get a lot of information about users and their browsing behavior, enabling such things as one-to-one marketing. Even more information is available to Internet service providers whose HTTP proxy servers may keep track of every Website visited by their subscribers. Similarly, it is difficult to publish data on the Web without revealing the corresponding HTTP server name or IP address. In this situation, privacy protection and anonymity services for the WWW are becoming increasingly important fields of study. This paper overviews and briefly discusses some technologies that are available today and that can be used to provide support for anonymous browsing and anonymous publishing on the WWW.

Keywords: World Wide Web; Privacy protection; Anonymity services

1. Introduction

Today, activities on the World Wide Web (WWW) are increasingly subject to observation. For example, as Web-based electronic commerce becomes more prevalent, the browsing behavior of Web users reveals individual shopping habits and spending patterns, as well as other data that people have traditionally considered to be personal and private. Similarly, the Web is becoming an important source for information gathering. In a competitive environment, a company may wish to protect its current research interests. However, monitoring HTTP traffic may indicate the company's primary focus. By keeping Web browsing characteristics private, the company's interests may be adequately protected, too. Finally, some electronic payment systems allow secure transactions over the Internet while preserving the untraceability that cash allows. However, if digital cash is transmitted over a channel that identifies both the purchaser and the merchant, the transaction may no longer stay anonymous.

In general, there are three types of anonymous communication properties that can be provided individually or combined [3]:

- Sender anonymity
- Receiver anonymity
- Unlinkability of sender and receiver (connection anonymity)

In short, sender anonymity means that the identity of the party who sent out a message

* Tel.: +41-31-325-96-96; fax: +41-31-325-90-30.
E-mail address: rolf.oppliger@bfi.admin.ch (R. Oppliger).

0167-739X/00/$ – see front matter
PII: S0167-739X(99)00062-X

The service is freely available to the public and accessible at URL http://janus.fernuni-hagen.de. It provides anonymity services for both browsers and Web publishers (servers):

- In order to provide anonymity services for the browser, the JANUS service acts as a proxy service. It accepts requests from arbitrary browsers, removes all data that may reveal information about the requesting user, and forwards the request to the server. Similarly, the server's response is relayed back to the browser. In this way, JANUS is conceptually similar to the Anonymizer and related services for anonymous browsing.
- In order to provide anonymity services for Web publishers and to support anonymous publishing accordingly, the JANUS service is able to encrypt and decrypt URLs on the fly in a way that these can be used as reference for a server. More precisely, if a request with an encrypted URL occurs, JANUS is able to decrypt the URL and forward it to the server, without enabling the user to get knowledge about the decrypted URL. Similarly, all references in the server's response are again encrypted before the response is forwarded to the browser. The feature of hiding the server's IP address or host name is the main advantage of using the JANUS service. In fact, it is the feature that supports anonymous publishing on the Web.

URL encryption and decryption is a suitable application for public key cryptography. Note that everybody should be able to encrypt and publish a URL, whereas only the JANUS service should be able to decrypt it. Basis of the encryption is the RSA algorithm which is used with 768-bit keys. For example, using JANUS to encrypt the URL http://www.ifi.unizh.ch/~oppliger results in the following expression:

http://janus.fernuni-hagen.de/janus_encrypted/MTA
mJIaWc+bdgumvf7i5xnsaYsQ6MTyg+VQnJpoHW3
+TDtb04ir$6gcAFlwdtEVrGhvNR8rSic2nbsK0D61$
3mqnJmi3LCY1lfT3gRN15yEOp EserUoAgy5i4LUk
VZccpWk=

Consequently, I could publish this expression in order to allow users to visit my homepage while staying anonymous. When a user requested this encrypted URL, JANUS would decrypt the request to see http://www.ifi.unizh.ch/~oppliger. It would then forward the request and return the result, encrypting and rewriting all of the URLs in the page so that your network logs show only a connection to JANUS, and even the URLs recorded in the logs do not reveal any information about the origin HTTP servers. On the other side, the server logs at `www.ifi.unizh.ch` only reveal that the homepage was accessed by `corona.fernuni-hagen.de` (or any other host that currently runs the JANUS service).

Nevertheless, there are at least three limitations and shortcomings that should be kept in mind when using the JANUS service:

- First, it is important to note that only the URLs are encrypted, so if somebody eavesdrops on the actual data stream, he will not be fooled.
- Secondly, the standard cautions about single-hop forwarding services apply: coordinated and well-placed sniffers could determine the mapping for a site.
- Thirdly, Webmasters must have a possibility to publish encrypted URLs.

The above-mentioned limitations and shortcomings of the JANUS service are addressed in a technology overviewed and briefly discussed next.

5.2. TAZ servers and the rewebber network

More recently, Ian Goldberg and David Wagner from the University of California in Berkeley have developed a more comprehensive and sophisticated approach to address the anonymous publishing problem [12]. The technology they propose can be viewed as a generalization of the JANUS service (similar to the fact that an anonymous remailer network generalizes the `anon.penet.fi` service). In fact, the researchers suggest the use of more (than just one) proxy servers to collectively disguise the server's location, and to encrypt the entire data stream between the requesting browser and the origin HTTP server. In addition, they also suggest the establishment of a directory service for encrypted URLs.

More precisely, the term 'rewebber' is used to refer to a proxy server that essentially implements the JANUS service's core functionality and the ability to encrypt the data traffic. In addition, each rewebber is

also able to understand 'nested' URLs (i.e., URLs of the form http://proxy.com/http://www.realsite.com). Most existing HTTP proxy servers have this ability. The basic idea is to publish a URL such as the one given above (which points to `proxy.com` instead of `www.realsite.com`), and to use public key cryptography to encrypt the real server name (the second part of the URL) so that only the rewebber can decrypt and actually see it. Encrypted URLs are also referred to as locators. In the example given above, the requested URL would look something like http://proxy.com/!RFkK4J... (`RFkK4J...` being the encrypted URL or locator for the URL http://www.realsite.com). The fact that the URL is encrypted is indicated by the leading ! instead of the expected `http://`. The rewebber at `proxy.com`, upon receiving the locator, would first decrypt it with its private key, and then proceed to retrieve the nested URL in the normal fashion. Obviously, this mechanism can be iterated to more than one rewebber. Consequently, all currently installed and operating rewebbers are collectively referred to as the rewebber network.

So far, the mechanism yet hides the real location of the origin HTTP server from the browser, but still has some flaws. First of all, once the browser has retrieved the resource from the rewebber, it could use one of the more powerful WWW search engines to try to find where the resource originally came from. This problem can be solved by encrypting the resource before storing it on the server. Thus, if the resource is accessed directly (e.g., through a WWW search engine), it will look like random data. In their implementation of a rewebber network, the researchers used the DESX encryption algorithm.[5] The DESX key is given to the rewebber in the encrypted part of the locator; that is, when the rewebber decrypts the locator, it finds not only a URL to retrieve, but also a DESX key with which to decrypt the resource thus retrieved. It then passes the decrypted resource back to the browser.

The technique of encrypting a resource stored at the server has another benefit: the resource can be padded in size before being encrypted, and the rewebber will truncate the resource to its original size before passing it back to the browser. The reason for this is similar to that for encrypting the resource in the first place. If the retrieved resource is 1234 bytes long, for example, a WWW search for encrypted resources near that size would quickly narrow down the possible choices. To thwart this, one can always add random padding to the end of encrypted resources so that their total length is one of a handful of fixed sizes, such as 10, 20, 40, 80, etc. kbytes.

To implement chaining, the URL in the encrypted portion of a locator is replaced with another (complete) locator, one which points to a rewebber at a different site (and preferably, in a different legal jurisdiction). As mentioned above, this process can be iterated, thus making the rewebber chain listed in the locator as long as one likes.

When using rewebber chaining, the resource will need to be multiply encrypted on the server. To do this, the publisher randomly selects a DESX key for each rewebber in the chain, encodes them into the locator, and iteratively encrypts the resource. In this way the publisher forms a locator and announces it in public (e.g., by using an anonymous remailer service to post it to a newsgroup). Note that all of the security parameters, including the length of the rewebber chain, are under the control of the publisher; individual publishers may adjust these parameters to fit their anonymity needs. The main benefit of chaining is that the rewebber closest to the browser ever sees the decrypted data, and only the rewebber closest to the server knows where it is really getting data from. In order to link the two, the cooperation of every rewebber in the chain would be necessary. This avoids the existence of a single point of failure, and allows the distribution of trust throughout the network.

Once a rewebber network is deployed, the stage will be set for the ability to publish anonymously on the Web. One major drawback, however, is that a locator that contains just a simple chain of rewebbers looks something like this:

http://rewebber.com/!RjVi0rawjGRT50ECKo_UBa
7Qv3FJIRbyej_Wh10g_9vpPAyeHmrYE1QL1H2ifN
h2Ma4UYt3laqeQRXXd7oxEvwR8wJ3cnrNbPF6rc1
Uzr6mxIWUtlgW0uRIL0bGkAv3fX8WEcBd1JPWG
T8VoY0F1jxgPL7OvuV0xtbMPsRbQg0iY=RKLBa
UYedsCn0N-UQ0m5JWTE1nuoh_J5J_yg1CfkaN9j
SGkdf51-gdj3RN4XHf_YVyxfupgc8VPsSyFdEeR0

[5] The DESX encryption algorithm refers to a technique intended to extend the strength of DES that was originally proposed by Ronald L. Rivest.

dj9kMHuPvLivE_awqAwU_3Af8mc44QBN0fMVJj
peyHSa79KdTQ5EGlPzLK7upFXtUFcNLSD7YLSc
1gKI3X8nk15s=RXbQaqm0Ax4VhKPwkLVK_MM
Jaz9wchn_pl48xhTzgndt5Hk09VToLyz7EF4wGH3X
KPD7YbKVyiDZytva_sUBcdqpmPXTzApYLBnl4n
DOy1o1Pu1R ky8CxRfnC9BvQqof853n99vkuGKP9
K4p3H7pl6i8DOat-NrOIndpz5xgwZKc=/

Obviously, this is hard to announce in public and there is a naming problem that needs to be solved (similar to the announcement of the encrypted URLs for the JANUS service).

In [12], it is proposed to create a virtual namespace called the `.taz` namespace (TAZ standing for 'Temporary Autonomous Zone'), and to create new servers called TAZ servers to resolve this namespace. The function of a TAZ server is to offer publishers an easy way to point potential readers at their material, as well as offering readers an easy way to access it. A TAZ server consists essentially of a public database mapping virtual hostnames ending in `.taz` to rewebber locators. Note that nothing in this database must be kept secret. Unlike a `anon.penet.fi`-style remailer (which associates a alias e-mail address with a real one), TAZ servers merely associate `.taz` addresses with locators. Most importantly, the TAZ server administrator cannot decrypt the locators that are stored in the database. There is the potential for a great deal of future work in the TAZ servers. Centralized solutions such as the one that was implemented so far may work well for a while, but in the future decentralized solutions may be preferable, for both scalability and availability reasons. A great deal could be learnt from the Internet's DNS. Anyway, more implementation and real-life deployment experience is needed to understand the engineering trade-offs better.

6. Conclusions

Although browsing on the Web feels like an anonymous activity, it is hardly that way. Website administrators generally get a lot of information about users and their browsing behavior, enabling such things as one-to-one marketing. Even more information is available to Internet service providers whose HTTP proxy servers may keep track of every Website visited by their subscribers. Similarly, it is difficult to publish data on the Web without revealing the corresponding HTTP server's name or IP address. In this situation, privacy protection and anonymity services for the WWW are becoming increasingly important fields of study. In this paper, we have overviewed and briefly discussed the technologies that are available today and that can be used for anonymous browsing and anonymous publishing on the WWW.

In addition to these technical approaches to provide anonymity services, there are also some voluntary cooperation privacy standards in development. For example, the W3C project Platform for Privacy Preferences (P3P) seeks to provide a platform for trusted and informed online interactions. The goal of P3P is to enable users to exercise preferences about Websites' privacy practices. P3P applications will allow users to be informed about Website practices, delegate decisions to their agent when they wish, and tailor relationships with specific sites [14]. It is assumed that users' confidence will increase when presented with meaningful choices about services and their privacy practices. Further information on P3P can be found at URL http://www.w3.org/P3P/.

Similarly, TRUSTe is an independent, non-profit organization dedicated to establishing a trusting environment where users can feel comfortable dealing with companies on the Internet. Championed by CommerceNet and the Electronic Frontier Foundation (EFF), TRUSTe's efforts focus on promoting trust through online privacy assurance, putting users in control of their personal information. Based on the principles of disclosure and informed consent, the TRUSTe Privacy Program, utilizes a branded, online 'seal' or trustmark to signify disclosure of a Website's personal information privacy policy. Sites that display the trustmark have formally agreed to adhere to the TRUSTe privacy principles, and to disclose their information gathering and dissemination practices. These companies must disclose what information they gather, how the information will be used, and who they share information with. To ensure that TRUSTe's privacy principles and the Website's disclosed practices are met, and the trustmark is not being misused, TRUSTe oversees a comprehensive assurance process. Further information on TRUSTe can be found at URL http://www.truste.org.

In summary, the handling of personal information on the Web is a hotly debated topic. The need to

maximize user's privacy is at odds at a fundamental level with businesses' need to minimize fraud. The first goal seeks to maximize users' anonymity, whereas the second goal requires users to be strongly and unequivocally identified and authenticated. Somehow, a compromise must be struck for this dilemma.

References

[1] R. Oppliger, Authentication Systems for Secure Networks, Artech House, Norwood, MA, 1996.

[2] D. Chaum, Untraceable electronic mail, return addresses and digital pseudonyms, Communications of the ACM 24 (2) (1981) 84–88.

[3] A. Pfitzmann, M. Waidner, Networks without user observability, Computers & Security 2 (6) (1987) 158–166.

[4] M.K. Reiter, Distributing trust with the Rampart toolkit, Communications of the ACM 39 (4) (1996) 71–74.

[5] M.K. Reiter, A.D. Rubin, Crowds: anonymity for web transactions, ACM TISSEC, vol. 1, 1998, p. 1.

[6] P.F. Syverson, M.G. Reed, D.M. Goldschlag, Private web browsing, Journal of Computer Security 5 (3) (1997) 237–248.

[7] C. Cülcü, G.Tsudik, Mixing emails with BABEL, Proc. ISOC SNDSS, February 1996, pp. 2–16.

[8] E. Gabber, P. Gibbons, Y. Matias, A. Mayer, How to make personalized web browsing simple, secure, and anonymous, Proc. Financial Cryptography, February 1997.

[9] M.G. Reed, P.F. Syverson, D.M. Goldschlag, Proxies for anonymous routing, Proc. ACSAC '96, IEEE Press, New York, December 1996, pp. 95–104.

[10] P.F. Syverson, D.M. Goldschlag, M.G. Reed, Anonymous connections and onion routing, Proc. IEEE Symposium on Security and Privacy, May 1997, pp. 44–54.

[11] D. Bleichenbacher, E. Gabber, P. Gibbons, Y. Matias, A. Mayer, On secure and pseudonymous client-relationships with multiple servers, Lucent Technologies, May 1998.

[12] I. Goldberg, D. Wagner, TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web, CS 268 Final Report, UC Berkeley, 1997.

[13] D. Kristol, L. Montulli, HTTP State Management Mechanism, RFC 2109, February 1997.

[14] M. Marchiori, D. Jaye, Platform for Privacy Preferences (P3P) Syntax Specification, W3C, July 1998, in preparation.

[15] R. Sandhu, J.S. Park, Cooking Secure Cookies on the Web, in preparation.

Rolf Oppliger studied computer science, mathematics, and economics at the University of Bern, Switzerland, where he received his M.Sc. and Ph.D. degrees both in computer science in 1991 and 1993, respectively. In January 1991, he also received the venia legendi for computer science from the University of Zurich, Switzerland. The focus of his current work and research activities is computer security in general, and network security in particular. He has authored and coauthored six books and several scientific papers and articles mainly on security-related topics. He is with the IT Security Group of the Swiss Federal Office of Information Technology and Systems (BFI) as well as the University of Zurich. He also serves as series editor for computer security for Artech House Publishers. Dr. Oppliger is a member of the Swiss Informaticians Society (SI) and its Working Group on Security, the Association for Computing Machinery (ACM), and the IEEE Computer Society. He also serves as vice-chair of IFIP TC 11/WG 4 on Network Security, and is representing Switzerland within the European Senior Officials Group on Security of Information Systems (SOG-IS) and the OECD Working Party on Information Security and Privacy (WISP).

is hidden, while its receiver and the message itself might not be. Similarly, receiver anonymity means that the identity of the receiver is hidden, while its sender and the message itself might not be. Finally, unlinkability of sender and receiver (also referred to as connection anonymity) means that though the sender and receiver can each be identified as participating in some communication, they cannot be identified as communicating with each other.

In the recent past, privacy protection and anonymity services for the WWW have become increasingly important fields of study. The aim of this paper is to overview and briefly discuss the technologies that are available today and that can be used for anonymous browsing and anonymous publishing on the WWW. The rest of the paper is organized as follows: Section 2 elaborates on previous work. Section 3 addresses the use of cookies and their implications for the privacy of Web users. Finally, Sections 4 and 5 address anonymous browsing and anonymous publishing, and Section 6 concludes with some final remarks.

2. Previous work

There is some previous work in providing anonymity services for electronic mail (e-mail) services. For example, `anon.penet.fi` was a simple and easy-to-use anonymous e-mail forwarding service (a so-called anonymous remailer) that was operated by Johan Helsingius in Finland. In short, the `anon.penet.fi` anonymous remailer was provided by an SMTP proxy server that stripped off all SMTP header information of each incoming e-mail message before forwarding it to its destination. Then, if not already assigned, an alias for the sender was created. In the outgoing message, the real e-mail address of the sender was replaced by the alias that allowed the recipient(s) of the message to reply to the sender without knowing his or her real identity or e-mail address. Consequently, `anon.penet.fi` provided sender anonymity by simply keeping the mapping between real e-mail addresses and their corresponding aliases anonymous. The downside of this simple approach was that any user of `anon.penet.fi` had to trust the service provider not to reveal his or her real identity or e-mail address. This level of trust may not always be justified. [1]

A more sophisticated approach to provide an anonymous e-mail forwarding service was proposed by David Chaum in the early 1980s [2]. In fact, Chaum introduced a technology that uses public key cryptography to provide anonymity services in a so-called Chaum mixing network. According to this terminology, a Chaum mix refers to an anonymous remailer that, in addition to forwarding incoming e-mail messages, strives to hide the relationship between incoming and outgoing data traffic. To achieve this, a Chaum mix typically encrypts the data traffic and may reorder, delay, and eventually pad data to eventually disable or complicate traffic analysis.

In a Chaum mixing network, the sender of an anonymous e-mail message first chooses a route through a series of mixes to the intended destination, and then wraps some extra layers of data around the message. To form the innermost layer, the name of the last mix M_n – the mix one hop away from the message destination – is concatenated with the original message, and the result is encrypted with the public key of the second-to-last mix M_{n-1} in the route. Consequently, the resulting bundle has one layer of routing data prepended to the original message, and it's encrypted with a key possessed only by M_{n-1}. If the bundle were to somehow arrive at M_{n-1}, it could be decrypted there, and the one layer of remaining routing data would be enough to get the message to M_n and from there to its final destination. This sort of message encapsulation can be repeated, the next time with the third-to-last mix M_{n-2}. The result is a bundle that can be decrypted only by M_{n-2}. Once decrypted there, the interior can be forwarded to M_{n-1}. At the same time, however, M_{n-2} cannot read the interior of the bundle, since that part is encrypted with the public key of M_{n-1} (of which the corresponding private key is known only by M_{n-1}).

One may think of this encapsulation scheme as an onion that is prepared by the sender. On the forward

[1] On 8 February 1995, based on a burglary report filed with the Los Angeles police, transmitted by Interpol, the Finish police presented Helsingius a warrant for search and seizure. Bound to do by law, he complied, thereby revealing the real e-mail address of a single user.

route to the destination, each mix peels off one layer of encryption.

If a Chaum mixing network were used to transmit e-mail messages only through one mix, this mix would have to be trusted not to reveal the senders' and receivers' identities (since it sees both of them). Consequently, most people forward e-mail messages through two or more mixes so no single one experiences both the sender and receiver of a particular e-mail message. In other words, using two or more mixes keeps the sender anonymous to every mix but the first and the receiver anonymous to every mix but the last. Consequently, a user's identity is best hidden if he actually runs his own Chaum mix and directs all of his outgoing e-mail messages through it.

If one is worried about an adversary powerful enough to monitor several Chaum mixes in a network simultaneously, then one also has to worry about timing and other correlation attacks. In the extreme case, suppose a Chaum mixing network is idle until a message is sent out, then even though an adversary cannot decrypt the layered encryption, he can still locate the route just by watching the right parts of the network and analyze the data traffic accordingly. Chaum mixing networks can resist such attacks with queues to batch, reorder, and process incoming messages. In fact, each mix may keep quiet – absorbing incoming messages but not transmitting them – until its outbound buffer starts to overflow, at which point the mix emits a randomly chosen message to its next hop. However, due to the real-time requirements of some applications, the batching, reordering, and processing of data in a queue is not always possible.

One question arises immediately with regard to the use of anonymous remailer services: how can the receiver of an (anonymous) e-mail message reply to the proper sender? The answer is that he cannot unless he is explicitly told how to do. A simple technique is to tell the receiver to send his reply to a certain newsgroup with a specific subject field. The reply can then be grabbed by the sender from the corresponding newsgroup. This approach of replying is yet untraceable but also expensive and unreliable. A more sophisticated technique would use the knowledge of how to build an untraceable forward route from the sender to the receiver, to build an inverse untraceable backward route from the receiver to the sender (note that the forward and backward routes are independent; they can be identical or completely disjunct). According to this technique, the sender computes a block of information that is used to anonymously return a reply from the receiver to the sender. This additional block of information is sometimes also referred to as a return path information (RPI) block. The RPI block is prepended to the original message and padding data that is sent from the sender to the receiver at first place.

The use of Chaum mixes to provide anonymous e-mail forwarding and RPI-based reply services was prototyped by Ceki Cülcü and Gene Tsudik at the IBM Zürich Research Laboratory. They used the scripting language Perl and the Pretty Good Privacy (PGP) software to build a system called BABEL [7].

Unfortunately, the lessons learnt from anonymous remailer services do not necessarily hold for WWW data traffic, since the characteristics of e-mail and Web-based applications are fundamentally different:

- First, the WWW is an interactive medium, while e-mail is store-and-forward;
- Secondly, e-mail is a 'push' technology, meaning that the sender of an e-mail message initiates the data transfer, possibly without even the knowledge or consent of the receiver (the existence of e-mail bombing attacks illustrates this point). By contrast, the WWW is a 'pull' technology, meaning that the receiver must explicitly request data from the sender.

The first difference implies that Chaum mixing networks are unacceptable (or at least difficult to use) for Web traffic. Nevertheless, the second difference also offers some possibilities to improve security (in terms of anonymity). For obvious reasons, the security of an anonymity-providing system, such as a Chaum mixing network, increases as the number of available and publicly accessible cooperating nodes (e.g., Chaum mixes) increases. In the realm of e-mail, operators of anonymous remailers often come under fire when their services are abused by people sending threatening letters or spam. In fact, the undesirability of handling irate users causes the number of anonymous remailers to stay considerably low, potentially impacting on the security of the overall system. By contrast, an HTTP or Web server cannot initiate a connection with an unwilling browser and send it data when no request was made. This 'consensual'

nature of the Web should cause fewer potential node operators to become discouraged, and therefore lead to corresponding increases in security. Finally, note that HTTP proxy servers are also well suited to implement anonymity services mainly because of their inherent caching capabilities (to improve network performance). The very fact that data is being cached at particular proxy servers makes it less likely that requests get forwarded all the way to the origin server. This makes traffic analysis more complicated and somehow harder to accomplish.

Contrary to anonymous e-mail forwarding services, there is only few research going on to provide anonymity services for the WWW. As a matter of fact, the consensus on WWW privacy protection is that there just is not much and that commercial interests are unlikely to champion the cause.

Before we delve into the details of anonymous browsing and anonymous publishing on the Web, we briefly discuss the use of cookies for HTTP state management.

3. Cookies

Let's assume a WWW server that should be configured to collect information about particular users to customize subsequent sessions.[2] In this situation, there are two possibilities:

- The server is configured to locally store the state information on a per-user basis;
- The server is configured to download the state information to the browsers where it is stored on the server's behalf.

Following the first approach, the server would have to build a huge database to store and make available state information related to particular users. This database tends to increase very rapidly. Contrary to that, the state information is not stored locally in the second approach. Instead, the information is downloaded to the browser where it is stored in a decentralized and fully distributed way. The next time, the browser connects to the server, it simply retransmits the appropriate state information (the one that 'belongs' to this particular server).

The HTTP state management mechanism as specified in RFC 2109 follows the second approach [13]. It uses the term 'cookie' to refer to the data string that encodes the state information that passes between the origin server and the browser, and that gets stored by the browser. More precisely, the mechanism specifies a way to create a stateful session with HTTP request and response messages. It describes two new headers, namely the Set-Cookie and the Cookie headers, that carry state information between participating origin HTTP servers and browsers.

To initiate a session, the origin HTTP server returns an extra response header to the browser, the Set-Cookie header. The browser, in turn, returns a corresponding Cookie header to the server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. The syntax and semantics of the Set-Cookie and Cookie headers are fully specified in the RFC mentioned above. For example, a typical Set-Cookie header would look as follows:

```
Set-Cookie: USER_NAME=Rolf;
path=/; expires=Wednesday,
18-Nov-99 23:12
```

The browser stores the cookie locally. If it requests a resource in path '/' on the same server (before 18 November 1999), it would send out the following Cookie header:

```
Cookie: USER_NAME=Rolf
```

Now the server knows that the requesting user has been previously assigned the username 'Rolf'. If there are other attributes being stored in the cookie, the server may also customize its behavior for this particular user.

The use of cookies raises some privacy concerns. In fact, a server may use Set-Cookie and Cookie headers to track the path of a user through the server. Users may object to this behavior as an intrusive accumulation of information, even if their identity may not be evident (identity may be evident if a user fills out a form that contains identifying information). Consequently, a user should be able to enable or disable the HTTP state management mechanism, and to either

[2] Note that the term 'session' here does not refer to a persistent HTTP connection but rather to a logical session created from HTTP request and response messages that belong together and correspond to each other.

reject or refuse cookies accordingly. This is possible in all major browsers, including Netscape Navigator and Microsoft Internet Explorer.

As of this writing, the HTTP state management mechanism is not cryptographically protected and must be considered to be unsecure. More recently, research has addressed possibilities to secure the HTTP state management mechanism with cryptographic technologies [15].

4. Anonymous browsing

In this section, we address four technologies that can be used to protect the privacy of Web users and to support anonymous browsing accordingly.

4.1. The Anonymizer

The Anonymizer[3] is a simple service that can be used to browse anonymously through the Web. As such, it is probably the most heavily used anonymizing service for the WWW. In short, the Anonymizer service is provided by a HTTP proxy server that runs on port 8080 of `www.anonymizer.com`. A user connects to this port, and the corresponding proxy server forwards the target URL to the origin HTTP server. In essence, the Anonymizer service is for the Web what the `anon.penet.fi` service was for e-mail: a simple and easy-to-use anonymous forwarding service. However, contrary to `anon.penet.fi`, the Anonymizer is a commercial service. You can either use this service for free (and pay a penalty in terms of a bonus 30–60 s delay per Web page and advertisements being included into the retrieved pages), or sign up for a paid account.

For example, if you want to use the Anonymizer service to retrieve the author's homepage located at URL http://www.ifi.unizh.ch/~oppliger, you type in http://www.anonymizer.com:8080/www.ifi.unizh.ch/~oppliger. This request will have the Anonymizer's HTTP proxy server retrieve the page from `www.ifi.unizh.ch` on your behalf. In doing so, the browser only leave traces originated from `sol.infonex.com` in the log files of the origin HTTP server (`sol.infonex.com` is the domain name system (DNS) name of the machine that currently hosts the Anonymizer service). Consequently, the administrator(s) of `www.ifi.unizh.ch` will not be able to reveal the IP addresses or DNS names of the systems that requested the above-mentioned URL using the Anonymizer service at first place.

In summary, the Anonymizer service is well suited to hide user identities and browser IP addresses from HTTP servers. Nevertheless, there are at least two problems that should be kept in mind and considered with care when using the service:

- First, a user has to trust the service provider not to reveal his identity. Note that the HTTP proxy server can be set up in a way that logs all requested URLs. Consequently, the Anonymizer service provider may get a considerably good picture about its users' browsing behavior on the Web. They have to trust the Anonymizer service provider not to reveal (or sell) this picture. Also, according to the Anonymizer user agreement, the logs are kept much longer than is technically required (15 days). So the Anonymizer can be raided just as `anon.penet.fi` was a couple of years ago.
- Secondly, although the Anonymizer is fine at hiding user identities and browser IP addresses from origin HTTP servers, it is not so good at hiding the server identities from the network segment(s) between the browser and the Anonymizer. For example, in the example given above the local network administrator (or the administrator of any proxy between the client and the Anonymizer service) can see `www.ifi.unizh.ch/~oppliger` being requested by just unpacking the URL.

Obviously, a possible solution to the first problem is to chain several anonymous HTTP proxy servers similar to the Anonymizer, whereas a solution to the second problem is to encrypt the target URL in a way that can be decrypted only by appropriate HTTP proxy servers (e.g., using the public keys of these servers).

4.2. Onion routing

More recently, a group of researchers at the US Naval Research Laboratory (NRL) have adapted the

[3] Anonymizer is a trademark of Anonymizer, The service provider can be contacted at URL http://www.anonymizer.com.

idea of using a Caum mixing network to implement anonymous connections. Anonymous connections are similar to TCP/IP connections, instead that they are resistant to both passive and active eavesdropping and traffic analysis attacks. Anonymous connections are bidirectional, have small latency, and can be used anywhere a TCP/IP connection can be used. Note that a connection may be anonymous, although communication need not be (e.g., if the data stream is not encrypted). The NRL researchers have prototyped the technology in a system called onion routing [6,9,10].[4]

In the descriptions that follow, we use the term 'onion' to refer to a layered encrypted message, whereas the term 'onion router' is used to refer to a Chaum mix that acts as forwarding node in an onion routing network.

In an onion routing network, instead of making TCP/IP connections directly to a responding machine (a responder), an initiating application (an initiator) establishes an anonymous connection through a sequence of onion routers. Contrary to normal routers, onion routers are connected by permanent and long-standing TCP/IP connections. Although the technology is called onion routing, the routing that occurs does so at the application layer (and not at the Internet layer). For any anonymous connection, the sequence of onion routers in a route is strictly defined at connection setup, and each onion router can only identify the previous and next hops along the route. Due to the use of encryption technologies, the data that passes along the anonymous connection appears differently at each onion router, so data cannot be tracked en route and compromised onion routers cannot collude.

In onion routing, an application does neither directly talk to a router nor to an onion router. Instead, there must be proxies that interface between the applications and the onion routing network. For example, to access a Website through an onion routing network, a user must set the browser's HTTP proxy to point to an onion network entry point (a so-called 'application proxy'). In fact, the initiator establishes a TCP/IP connection to an application proxy, and this proxy defines a perhaps random route through the onion routing network by constructing a layered data structure (an onion) and sending that onion through the network. Similar to a Chaum mixing network, each layer of the onion is encrypted with the public key of the intended onion router and defines the next hop in the route. An onion's size is fixed, so each onion router adds some random padding data to replace the removed layer. The last onion router forwards data to the responder's application proxy, whose job is to pass data between the onion routing network and the responder. In addition to carrying next hop information, each onion layer also contains key seed material from which cryptographic keys are derived (for encrypting and decrypting data sent forward and backward on the route of the anonymous connection).

After having sent the onion, the initiator's application proxy starts sending data through the anonymous connection. As data moves through the anonymous connection, each onion router removes one layer of encryption, so it finally arrives as plaintext. Obviously, the layering occurs in the reverse order for data moving backward to the initiator. Stream ciphers are used for data encryption and decryption. Similar to the original idea of a Chaum mixing network, onion routers may also randomly reorder the data they receive before forwarding it (but preserve the order of data in each anonymous connection).

As mentioned previously and contrary to the original idea of a Chaum mixing network, the batching technique is out of the question for the support of interactive applications, such as Web browsing. This means that coordinated observation of the network links connecting onion routers could reveal an anonymous connection's route and reveal the source and destination IP addresses accordingly. Therefore, it is important to ensure that the links between the onion routers cannot be simultaneously eavesdropped. The easiest approach is to put onion routers on different network segments in different buildings with different administrators – ones who would be unlikely to collude. Also note that by layering cryptographic operations in the way described above, an advantage is gained over plain old link layer encryption. Even through the total cryptographic overhead for passing data is the same as for link layer encryption, the protection is better. In link layer encryption, the chain is as strong as the weakest link: one compromised node can reveal everything.

[4] The onion routing system is conceptually similar to the PipeNet proposal that was posted by Wei Dai to the Cypherpunks mailing list in February 1995. Contrary to the onion routing system, the PipeNet proposal has not been implemented so far.

In onion routing, however, the chain is as strong as its strongest link: one honest onion router is enough to maintain the anonymity of the connection. Even if link layer encryption is used together with end-to-end encryption, compromised nodes can cooperate to reveal route information. This is not possible in an onion routing network, since data always appears differently to each onion router.

For TCP-based application protocols that are proxy-aware, such as HTTP, Telnet, and SMTP, there exist proxies for Sun Solaris. Surprisingly, for certain applications that are not proxy-aware, most notably `rlogin`, it has been possible to design interface proxies, as well. In either case, the best protection results from having a connection between an application proxy and an onion router that is trusted on one way or another. For example, one possibility is to place an onion router on the firewall of a Website. In this case, the onion router would serve as an interface between the machines behind the firewall and the external network (most notably the Internet). This firewall configuration is commonly used today.

Refer to http://www.onion-router.net for current status and other useful information about onion routing.

4.3. Lucent personalized web assistant

An increasingly number of Websites require users to establish an account before they can access the site. This approach is sometimes called 'personalized Web browsing.' Typically, the user is required to provide a unique username, a password, and an e-mail address.

Establishing accounts at multiple sites is generally a tedious task. A user may have to invent a distinct username and a secure password, both unrelated to his identity, for each Website. In addition, the user may also want support for anonymous e-mail. Besides the information that the user supplies voluntarily to the Website, additional information about the user may flow involuntarily from the user's browser to the Website, due to the nature of HTTP and the use of cookies (as described in Section 3).

Against this background, a group of researchers at Lucent Technologies has developed a technology that makes personalized Web browsing simple, secure, and anonymous by providing convenient solutions to each of the problems mentioned above [8]. The technology has been implemented in a system called Lucent Personalized Web Assistant (LPWA). In short, the LPWA is an agent that may interact with several Websites on the user's behalf. It automatically derives a unique pseudonym or alias for a user at each site he visits, and transparently presents that alias to the site on request. Typically, the alias consists of a username, a password, and eventually an e-mail address. In general, different aliases are generated for each user and Website pair, but the same alias is presented whenever a user visits a particular Website.

Providing support for personalized Web browsing, the LPWA frees the user from the burden of inventing and memorizing distinct usernames and passwords for each Website, and guarantees that an alias (including an e-mail address) does not reveal the identity of the user. In addition, the LPWA also provides support for a Website to reply to anonymous e-mail messages originated by a particular user. Finally, the LPWA is also able to filter the HTTP data traffic to preserve user privacy. As such, the developers of LPWA claim that their system 'provides simultaneous user identification and user privacy, as required for anonymous personalized Web browsing.'

At the core of anonymous personalized Web browsing is the problem of names translation: translating from the user's e-mail address and secret to an alias that fulfils a number of properties, including anonymity, consistency, secrecy, uniqueness of an alias, and protection from creation of dossiers. To address these requirements, a specific collision-resistant one-way hash function is used, and this function is called a Janus function. Input to the Janus function are the real LPWA username and password as well as the name of the requested Website [11].

In practice, the LPWA can be configured as a remote server (central proxy), as a local server (local proxy), or anywhere in between (e.g., firewall proxy), with different trade-offs in terms of security, trust, and convenience. The configuration of the current demonstration is a single copy of LPWA running on a proxy at port 8000 of `lpwa.com`. To use this central proxy, one must first configure the browser to automatically use it. The basic idea is to set the browser's HTTP proxy to `lpwa.com` port 8000. Consider the fact that you want to register and access a Website of a company that requires the submission of a username and

password. LPWA can automatically generate this information, and even a unique e-mail address where you can receive return mail. For example, after setting your browser to use the LPWA proxy, you can go to the home page of the company. LPWA will interpose a couple of pages describing the LPWA service and asking you for your real identity (or at least the one from which your pseudonyms will be derived). Then when LPWA sends you on to your original URL and the Website asks you to register, just give the string \u as your username, \p as your password, and @ as your e-mail address. LPWA will intercept those codes and replace them with the nonsensical pseudonyms it uses for you at that particular site. Log entries in the origin HTTP server will then reveal requests originated from `lpwa.com` only.

4.4. Crowds

A group of researchers at AT&T Research has developed a system called Crowds for protecting users' anonymity on the WWW [5]. Unlike other technqiues addressed so far, Crowds does not rely on Chaum mixes at all. Instead, Crowds, named for the notion of 'blending into a crowd', operates by grouping users into a large and geographically diverse group (a so-called 'crowd') that collectively issues requests on behalf of its member users. As such, Crowds is essentially a distributed and chained Anonymizer, with encrypted links between individual Crowds members. HTTP data traffic is forwarded to a crowd member, who flips a biased coin and, depending on the result, forwards it either to some other crowd member or to the origin HTTP server. This makes communication resistant to observers.

More precisely, a crowd can be thought of as a collection of users. Each user is represented in a crowd by a process that runs on his system. In Crowds parlance, this process is called a jondo (pronounced 'John Doe' and meant to convey the image of a faceless participant). The user or a system administrator acting on the user's behalf starts the jondo. When it is started, it contacts a server called the blender to request admittance to the crowd. If admitted, the blender reports to the jondo the current membership status of the crowd and information that enables the jondo to participate in the crowd. The user, in turn, configures the jondo to serve as HTTP proxy by specifying its hostname and port number in his browser for all services, including Gopher, HTTP, and SSL.

Thus, any request originating from the browser is sent directly to the jondo. Upon receiving the first request from the browser, the jondo initiates the establishment of a random path of jondos to and from the origin HTTP server. More precisely, the jondo picks a jondo from the crowd (possibly itself) at random, and forwards the request to it. When this jondo receives the request, it flips a biased coin to determine whether or not to forward the reuqest to another jondo. If the result is to forward, then the jondo selects a random jondo and forwards the request to it. Otherwise the jondo submits the result to the HTTP server for which the request was destined. Consequently, each request travels from the user's browser, through a number of jondos, and finally to the origin HTTP server. Subsequent requests initiated at the same jondo follow the same path (except perhaps going to a different HTTP server), and server response messages traverse the same path as the request messages, only in reverse.

All communication between any two jondos is encrypted using a key known only to the two of them. Encryption keys are established as jondos join the crowd. Therefore, some group membership procedures must be defined. Those procedures determine who can join the crowd and when they can join, and inform members of the crowd membership accordingly. In fact, there are many schemes and corresponding group membership protocols that could be used to manage crowd memberships. While providing robust and reliable distributed solutions, many of these schemes have the disadvantage of incurring significant overhead and of providing semantics that are too strong for the application at hand. In the Crowds system, a simpler and centralized solution is used. As already mentioned above, membership in a crowd is controlled and reported to crowd members by the blender. To make use of the blender (and thus the crowd), the user must establish an account with the blender, i.e., an account name and password that the blender stores. When the user starts up a jondo, the jondo and the blender use this shared secret (the password) to authenticate each other's communication. As a result of this communication, the blender may accept the jondo into the crowd and add the jondo (i.e., its IP address, port number, and account name) to its current list of members, as

well as reports this list back to the jondo. In addition, the blender also generates and reports back a list of shared keys, each of which can be used to authenticate another member of the crowd. The blender then sends each key to the other jondo that is intended to share it (encrypted under the account password for that jondo) and informs the other jondo of the new member. At this point all members are equipped with the data they need for the new member to participate in the crowd.

Each member maintains its own list of crowd memberships. This list is initialized to that received from the blender when the jondo joins the crowd, and is updated when the jondo receives notices of new or deleted members from the blender. The jondo can also (autonomously) remove jondos from its list of crowd members, if it detects the corresponding jondos have failed. This allows for each jondo's list to diverge from others' if different jondos have detected different failures in the crowd.

Obviously, a major disadvantage of this centralized approach to group membership management is that the blender is a trusted third party (TTP) for the purposes of key distribution and membership reporting. Techniques exist for distributing trust in such a TTP among many replicas, in a way that the corruption of some fraction of the replicas can be tolerated [4]. In its present, non-replicated form, however, the blender is best executed on a trusted computer system (e.g., with login access available only at the console). Note, however, that even though the blender is ATTP for some functions, HTTP traffic is not generally routed through the blender, and thus a passive attack on the blender does not immediately reveal the users' Web transactions. Moreover, the failure of the blender does not interfere with ongoing transactions. It is planned that in future versions of Crowds, jondos will establish mutually shared keys using the Diffie–Hellman key exchange, where the blender serves only to authenticate and distribute the Diffie–Hellman public values of the Crowds members. This will eliminate the present reliance on the blender for key generation. Another possibility would be the use of Kerberos or another authentication and key distribution system [1].

A thorough security and performance analysis for Crowds is given in [5]. Crowds 1.0 is implemented in Perl 5.0. According to their developers, this scripting language was chosen for its rapid prototyping capabilities and its portability accross Unix and Microsoft platforms. While Crowds performance is already encouraging, it could be further improved by reimplementing the system in a compiled language, such as C and C++. Further information on Crowds and the corresponding software can be obtained from the project's homepage that is located at URL http://www.research.att.com/projects/crowds. Note, however, that due to US export restrictions, the software can be obtained by US and Canadian citizens only.

5. Anonymous publishing

The technologies overviewed and discussed so far address the problem of how to protect the privacy of Web users, and how to provide support for anonymous browsing. Contrary to that, this section addresses the problem of how to publish data anonymously on the Web. Note that the current WWW architecture provides little support for anonymous publishing. In fact, the architecture fundamentally includes information in the URL that is used to locate resources, and it is very hard for a Web publisher to avoid revealing this information (at least if it is required that resources published anonymously be accessible from standard Web browsers without needing any special client software or anonymity tool). Also note that the browser privacy problem is orthogonal to the anonymous publishing problem, and that the two problems compose well: if full anonymity is needed, techniques for anonymous browsing will work well in tandem with an infrastructure for anonymous publishing.

In the subsections that follow, two basic technologies are presented that can be used to address the problem of anonymous publishing on the Web. The first technology is rather simple and straightforward, whereas the second technology is more complex and sophisticated.

5.1. JANUS

JANUS is a joint research project of the Forschungsinstitut für Telekommunikation (FTK) of Dortmund, Hagen, and Wuppertal in Germany. One of the results of the project is an anonymous publishing service that is currently provided by the Fernuniversität Hagen.